

**UNITED STATES PATENT APPLICATION FOR:**

**CACHE LINE PURGE AND UPDATE INSTRUCTION**

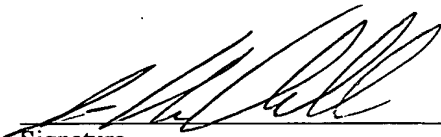
**INVENTORS:**

**STEVEN R. KUNKEL  
DAVID ARNOLD LUICK**

**ATTORNEY DOCKET NUMBER: ROC920010209US1**

**CERTIFICATION OF MAILING UNDER 37 C.F.R. 1.10**

I hereby certify that this New Application and the documents referred to as enclosed therein are being deposited with the United States Postal Service on January 22, 2002, in an envelope marked as "Express Mail United States Postal Service", Mailing Label No. EV041915992US, addressed to: Assistant Commissioner for Patents, Box PATENT APPLICATION, Washington, D.C. 20231.

  
Signature

Gero G. McClellan

Name

January 22, 2002

Date of signature

202201240500F

## **CACHE LINE PURGE AND UPDATE INSTRUCTION**

### **BACKGROUND OF THE INVENTION**

#### **Field of the Invention**

[0001] The present invention generally relates to multiple processor shared memory computer systems and more specifically to managing cache memory in such systems.

#### **Description of the Related Art**

[0002] Users of data processing systems continue to demand greater performance for handling increasingly complex and difficult tasks. Greater performance from the processors that operate such systems may be obtained through faster clock speeds so the individual instructions are processed more quickly. However, processing speed has increased much more quickly than the speed of main memory. Despite the speed of a processor, a bottleneck on computer performance is that of transferring information between the processor and memory. Therefore, cache memories, or caches, are often used in many data processing systems to increase performance in a relatively cost-effective manner.

[0003] A cache is typically a relatively faster memory that is intermediately coupled between one or more processors and a bank of slower main memory. Cache speeds processing by maintaining a copy of repetitively used information in its faster memory. Whenever an access request is received for information not stored in cache, the cache typically retrieves the information from main memory and forwards the information to the processor. If the cache is full, typically the least recently used information is discarded or returned to main memory to make room for more recently accessed information.

[0004] The benefits of a cache are realized whenever the number of requests to address locations of cached information (known as "cache hits") are maximized relative to the number of requests to memory locations containing non-cached information (known as "cache misses"). Despite the added overhead that occurs as a result of a

cache miss, as long as the percentage of cache hits is high (known as the "hit rate"), the overall processing speed of the system is increased.

[0005] Illustratively, one method of increasing the hit rate for a cache is to increase the size of the cache. However, cache memory is relatively expensive and is limited by design constraints, particularly if the cache is integrated with a processor on the same physical integrated circuit.

[0006] As an illustration, one cost-effective alternative is to chain together multiple caches of varying speeds. A smaller but faster primary cache is chained to a relatively larger but slower secondary cache. Furthermore, instructions and data may be separated into separate data and instruction caches. Illustratively, some processors implement a small internal level one (L1) instruction cache with an additional external level two (L2) cache, and so on.

[0007] Shared-memory multiprocessor systems present special issues regarding cache implementations and management. In shared-memory multiprocessor systems, all processors can access all memory including main and cache memory. This enables the tasks on all of the processors to efficiently and easily share data with one another. However, this sharing must be controlled to have predictable results. Conventionally, share-memory multiprocessor systems have hardware that maintains cache coherence and provides software instructions that can be used to control which processor is writing to a particular memory location. In order to prevent multiple processors from storing to the same memory location (or cache line) at the same time, most shared memory multiprocessors use a snoop-invalidate cache protocol to allow a processor to write data to a memory location (or cache line) only if it has an exclusive copy of the cache line containing the memory location.

[0008] In a system with a large number of processors, the next processor to read and/or write to a memory location is often not the processor that has the cache line stored in the cache associated with that processor. This requires the cache line to be moved between the caches of different processors. Efficiently moving cache lines to other caches is critical to multiprocessor systems.

[0009] On a shared-memory multiple processor system with 16 megabytes of level two (L2) cache per processor, about forty percent of the cache misses are due to reading and/or writing of shared data. Making the cache larger or adding additional levels of cache does not reduce the amount of cache misses. Instead, the result is the percentage of cache misses become larger with a larger cache and movement of the cache lines between caches reduces the performance of multiple processor systems.

[0010] Therefore, there is a need for a mechanism that will reduce the amount of cache misses in shared-memory multiple processor systems and improve overall system performance.

## SUMMARY OF THE INVENTION

[0011] The present invention generally provides a method and apparatus for purging a cache line and sending the purged cache line to the cache of one or more other processors in a shared-memory multiple processor computer system.

[0012] In one embodiment, a cache line purge instruction configures a processor to purge a cache line from the processor and sends the cache line to at least one of a plurality of processors in the computer system to update the at least one of a plurality of processors.

[0013] In another embodiment, the cache line purge instruction, when executed by a processor, updates all processors in the computer system.

[0014] In yet another embodiment, the cache line purge instruction updates only the oldest cache line in the computer system.

[0015] In still another embodiment, the cache line purge instruction updates only one cache at a designated processor, marks the updated cache line state as exclusive and marks the state of the cache of the processor executing the instruction as temporarily invalid.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

[0016] So that the manner in which the above recited features, advantages and objects of the present invention are attained and can be understood in detail, a more particular description of the invention, briefly summarized above, may be had by reference to the embodiments thereof which are illustrated in the appended drawings.

[0017] It is to be noted, however, that the appended drawings illustrate only typical embodiments of this invention and are therefore not to be considered limiting of its scope, for the invention may admit to other equally effective embodiments.

[0018] Figure 1 is a block diagram of a computer system consistent with the invention.

[0019] Figure 2 illustrates a diagram of a cache directory.

[0020] Figure 3 illustrates an instruction used to purge a cache line from one processor and provide the cache line to another processor.

[0021] Figure 4 shows a flow diagram of the method for purging and updating a cache line.

## **DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT**

[0022] The present invention provides a processor instruction that purges a cache line from an issuing processor and sends the purged cache line to the cache of one or more other processors in a shared-memory multiple processor system. The instruction enables the data contained in the purged cache line to be moved to one or more other processors before the other processor(s) need the data.

[0023] One embodiment of the invention is implemented as a program product for use with a computer system such as, for example, the computer system show in FIG. 1 and described below. The program(s) of the program product defines functions of the embodiments (including the methods described below with reference to FIGs. 2, 3 and 4) and can be contained on a variety of signal/bearing media. Illustrative signal/bearing

media include, but are not limited to: (i) information permanently stored on non-writable storage media (e.g., read-only memory devices within a computer such as CD-ROM disks readable by a CD-ROM drive); (ii) alterable information stored on writable storage media (e.g., floppy disks within a diskette drive or hard-disk drive); or (iii) information conveyed to a computer by a communications medium, such as through a computer or telephone network, including wireless communications. The latter embodiment specifically includes information downloaded from the Internet and other networks. Such signal-bearing media, when carrying computer-readable instructions that direct the functions of the present invention, represent embodiments of the present invention.

[0024] In general, the routines executed to implement the embodiments of the invention, may be implemented as part of an operating system or a specific application, component, program, module, object, or sequence of instructions. The inventive computer program is typically comprised of a multitude of instructions that will be translated by the native computer into a machine-readable format and hence executable instructions. In addition, programs are comprised of variables and data structures that either reside locally to the program or are found in memory or on storage devices. In addition, various programs described hereinafter may be identified based upon the application for which they are implemented in a specific embodiment of the invention. However, it should be appreciated that any particular program nomenclature that follows is used merely for convenience, and thus the invention should not be limited to use solely in any specific application identified and/or implied by such nomenclature.

[0025] Referring now to Figure 1, a computer system 110 consistent with the invention is shown. For purposes of the invention, computer system 110 may represent any type of computer, computer system or other programmable electronic device, including a client computer, a server computer, a portable computer, a minicomputer, a midrange computer, a mainframe computer, an embedded controller, etc. adapted to support the methods, apparatus and article of manufacture of the invention. The computer system 110 may be a standalone device or networked into a larger system. In one embodiment, the computer system 110 is an eServer iSeries computer available from International Business Machines of Armonk, New York.

[0026] The computer system 110 could include a number of operators and peripheral

systems as shown, for example, by a mass storage interface 136 operably connected to a direct access storage device 138, by a terminal interface 140 operably connected to a terminal 142, and by a network interface 144 operably connected to a plurality of networked devices 146. The terminal 142 and networked devices 146 could be desktop or PC-based computers, workstations, network terminals, or other networked computer systems.

[0027] Illustratively, computer system 110 is shown for a multi-user programming environment that includes at least two processors, processor 112 and processor 124, which obtains instructions, or operation codes, (also known as opcodes), and data via a bus 130 from a shared main memory 116. Illustratively, each processor has its own cache and each cache may be coupled to a plurality of successive caches. Successive caches may be labeled as numerically sequential levels of cache such as level one (L1), level two (L2), etc. as they extend from the processor. As an illustration, processor 112 is coupled to cache 114 and processor 124 is coupled to cache 126. Although shown as single entities, the caches 114 and 124 are representative of any number of levels of cache. In general, the processors 112 and 124 may be any processors configured to support the aspects of the present invention. In particular, the computer processors 112 and 124 are selected to support monitoring of processor cache. Illustratively, the processors are PowerPC processors available from International Business Machines, Inc. of Armonk, New York.

Sub  
A1  
10054042-012202  
[0028] The main memory 116 could be one or a combination of memory devices, including Random Access Memory, nonvolatile or backup memory, (e.g., programmable or Flash memories, read-only memories, etc.). In addition, memory 116 may be considered to include memory physically located elsewhere in a computer system 110, for example, any storage capacity used as virtual memory or stored on a mass storage device or on another computer coupled to the computer system 110 via bus 128.

[0029] Illustratively, the main memory 116 includes an operating system 118, and a computer program 120. As an illustration, the operating system 118 contains a cache purge instruction 128 which, when executed, purges a cache line from the cache of one of the processors in computer system 110 and sends the cache line to one or more other processor in computer system 110. Although the instruction 128 is shown as a

part of the operating system 118, the instruction 128 may be contained in the computer program 120 or in any other executable program.

[0030] Figure 2-illustrates one embodiment of a cache directory 200. In some embodiments, the cache directory 200 may be configured as is known in the art. In general, the cache directory 200 contains information about which caches contain which memory locations. A cache line comprises data stored in a cache referenced by a cache directory entry. The cache directory is queried before any change to a cache line is initiated. When an entry in the cache is changed or modified, the directory updates, invalidates or temporarily invalidates the respective directory entry corresponding to the cache line.

[0031] Figure 2 illustrates a cache line 202 comprising a cache directory and data 210. As an illustration, the cache directory 200 comprises a tag 204, replacement information 206, state 208 and a processor history 212. The tag 204 is a portion of the address in main memory 116 that is used to identify the data 210 stored in main memory 116. The replacement information 206 contains historical information about the age of the cache line 202. In one embodiment, the replacement information 206 could be the amount of system processing cycles that have elapsed since the last time the cache line was accessed by the system. In another embodiment, the replacement information 206 could be a system time stamp indicating when the cache line was first created or last modified. The state field 208 indicates the status of the cache line 202. Illustratively, the state of the cache line 202 could be modified, exclusive, shared or invalid. A modified state indicates that the data 210 referenced by the tag 204 has been changed. An exclusive state indicates that the memory location exists in only one cache and can be modified by the corresponding processor. A shared state indicates that multiple copies of the memory location may exist. An invalid state indicates that the cache line 202 no longer contains current useful information and may be overwritten by a new cache line entry. The processor history 212 contains information identifying the last processor that previously used the cache line. In one embodiment, the information may be any numerical or alpha numerical data used to uniquely identify a processor.

[0032] The method for purging and updating cache lines comprises a state called "temporarily invalid" that may be entered into the state field 208 of the cache directory



200. Illustratively, the temporarily invalid state indicates that the cache line 202 once held the memory location indicated by the tag 204 but is currently being accessed exclusively by another processor that requested the cache line 202 exclusive. While the cache line 202 is in a temporarily invalid state, the cache line will not be overwritten by a new cache entry. A new cache entry will first replace a cache line that is in an invalid state. If no cache lines are in an invalid state, the new cache entry will replace the oldest cache line 202 as indicated by the historical information contained in the replacement information 206. The temporarily invalid cache line will only be overwritten if the replacement information 206 indicates that the temporarily invalid cache line is the oldest cache line.

Sub  
P2  
10054042-012202

[0033] Figure 3 illustrates one embodiment of a cache purge instruction 128 used to purge a cache line addressed in field "RA" 310. Illustratively, the cache purge instruction 128 is an X-form PowerPC instruction 128 and the opcode 302 may be any notation recognized by a processor to execute the instruction. For example, the opcode 300 may be represented by a hexadecimal notation. The field "M" contains information regarding which processor(s) and their associated caches will be updated and may contain a value equal to "0" or "1". For example, if the field "M" 304 equals "0", then the processor and its associated cache referenced by a processor number stored in field "RB" 312 is updated. Further, if the processor number stored in field "RB" 312 is the processor that is executing the instruction 128, then all processors and their associated caches are updated. If field "M" 304 equals "1", the processor chosen to be updated is determined based on the processor history 212 shown in Figure 2. Referring back to Figure 3, the field "H" 308 indicates the cache level that is to be updated and may contain any numerical value representative of a level of cache. As an illustration, if field "H" contains "0" then the level one (L1) cache is updated and so on. The field "P" 306 contains a value that indicates under what circumstances each processor in the system performs an update write to its cache or caches while updating the state field 208 of the cache line 202 in the cache directory 200. As an illustration, if field "P" 306 equals "0", then all caches are updated and the state field 208 of the cache line addressed in field "RA" 310 is marked as shared. If field "P" equals "1", then the state of only one cache line at a designated processor is updated and marked exclusive while the issuing processor marks the cache line temporarily invalid. If field "P" equals "2" then all caches are updated and the cache line state is marked as temporarily invalid.

[0034] One embodiment illustrating a method of purging and updating a cache line is shown as a method 400 in Figure 4. In one embodiment, the method 400 may be understood as illustrative of the operation of the system 110. The method 400 is entered at step 402 whenever the instruction 128 is executed. At step 402, the method queries if field "P" 306 equals "1" indicating that the state of only one cache line at designated processor is to be updated. If so, the method proceeds to step 404 where the method queries if field "M" 304 equals "1" indicating that the processor chosen to be updated is determined based on the processor history 212 shown in Figure 2. If so, the method 400 proceeds to step 406 where the system hardware is used to determine which processors to update based on processor history 212. This is accomplished by examining the processor history 212 in the cache directory 200 to find the processors that previously used the cache line. In one embodiment, only one processor is selected to be updated. For instance, the processor referenced to the oldest cache line is selected as the processor to be updated. In an alternative embodiment, more than one processor may be selected by hardware to be updated. The method then proceeds to step 407 where a decision is made as to whether only one processor is to be updated. If only one processor is to be updated, the method then proceeds to step 426 where an update command along with updated data are sent to the processor determined to be updated. In one embodiment, if there is no processor or if there is more than one processor selected to be updated, the method proceeds to step 416. In an alternative embodiment, if there is no processor or if there is more than one processor selected to be updated, the method proceeds to step 424. At step 432, the processor determined to be updated updates its cache contents and marks the cache line exclusive. This is because the processor to be updated is the only processor that has this cache line to be updated. At step 438, the processor executing the instruction 128, marks its cache line temporarily invalid.

[0035] If, at step 404, the answer to the query is in the negative meaning that field "M" equals "0, the method then proceeds to step 418 where the method 400 queries if field "RB" 312 equals the processor executing the instruction 128. If not, the method proceeds to step 420 where only the processor contained in field "RB" 312 is updated. The method 400 then proceeds to step 426. If at step 418 it is found that the "RB" 312 equals the processor executing the instruction 128, then the method proceeds to step 424 where an update command along with updated data is sent to all processors. At

step 430, all processors update their caches at all levels and mark the cache line as shared. At step 436, the processor executing the Instruction 128 marks its cache line as shared.

[0036] If at step 402, field "P" 306 does not equal "1", then the method 400 proceeds to step 412 where the method queries if field "P" 306 equals "0". If so, the method proceeds to step 424, which has been described above. If not, the method proceeds to step 410 where the method 400 queries if field "P" 306 equals "2". If so, the method proceeds to step 416 where an update command along with updated data is sent to all processors. At step 422, each processor in the system looks at the state of the cache line in all of their cache levels. At step 428, the method 400 queries if the state of the cache line is marked as temporarily invalid. If so, the method 400 updates the contents of the cache line marked temporarily invalid and marks the line as shared. The method then proceeds to step 436.

[0037] In one embodiment, a processor may execute the instruction 128 after the processor executes at least one store instruction. The value of field "M" 304, field "P" 306, field "H" 308, field "RA" 310, and field "RB" 312 may be selected to ensure that other processors have a copy of the data of the store instruction.

[0038] For instance, assume a first processor executes a store instruction to put data 1234H to memory address [56789H] (square brackets indicate the enclosed number is a memory address. Assume further it is known that a second processor will soon reads data from memory address [56789H]. To improve the performance of the second processor, the first processor may execute an instruction 128 after the first processor executes the store instruction. For the instruction 128, field "RB" 312 can hold the processor number of the second processor. Field "M" 304 can be 0 so that the second processor whose processor number is in field "RB" 312 is selected to be updated. Field "P" 306 can be 1 so that only the second processor is selected to be updated. Field "H" 308 can be 0 so that an L1 cache of the second processor is updated.

[0039] Relating the method of Fig. 4 to the above example, in step 402, because field "P" 306 is 1, the method proceeds to step 404. In step 404, because field "M" 304 is 0, the method proceeds to step 418. In step 418, because field "RB" 312 contains the

processor number of the second processor, not the issuing, first processor, the method proceeds to step 420. In step 420, only the second processor is updated. In step 426, a command and data 1234H are sent to the second processor via bus 130. In step 432, the second processor updates a cache line of its L1 cache for the memory address [56789H] and marks the cache line Exclusive. In step 438, the first processor marks as Temporary Invalid its cache line for memory address [56789H]. As a result, when the second processor reads the most current data 1234H from memory address [56789H], the data is present in its L1 cache and a cache read miss is avoided. Therefore, performance of the second processor is improved leading to performance improvement of the entire system 110.

[0040] While the foregoing is directed to embodiments of the present invention, other and further embodiments of the invention may be devised without departing from the basic scope thereof, and the scope thereof is determined by the claims that follow.

10054042 012202